Making Tuple Spaces Physical with RFID Tags

Marco Mamei, Renzo Quaglieri, Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria – Università di Modena e Reggio Emilia
Via Allegri 13 – 42100 Reggio Emilia – Italy
name.surname@unimore.it

ABSTRACT
In this paper, we describe the design and implementation of a tuple-based distributed memory realized with the use of RFID technology. The key idea – rooted in a more general scenario of pervasive and mobile computing – is that our everyday environments will be soon pervaded by RFID-tagged objects. By accessing in a wireless way the re-writable memory of such RFID tags according to a tuple-based access model, it is possible to enforce mobile and pervasive coordination and improve our interactions with the physical world. An application example is presented to outline the potential of the approach.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features. C.2.4 [**Computer-Communication Systems**]: Distributed Systems; C.3 [**Special Purpose and Application-based Systems**]: Smartcards.

General Terms
Algorithms, Design

Keywords
Pervasive Computing, Tuple-based Coordination, RFID Tags.

1. INTRODUCTION
Tuple-based coordination models [6], providing distributed agents with associative content-based access to shared dataspaces (i.e., tuple spaces), well suit modern distributed computing scenarios. In fact, tuple spaces may act both as general repositories of shared contextual information and as suitable means to support uncoupled coordination between agents.

While tuple-based coordination has already been extensively adopted for the coordination of distributed agents in distributed computational environments (e.g., high-performance computing [1] and Internet computing [9]), the next challenge is to take advantage of emerging pervasive computing technologies to make tuple spaces useful to improve our coordination in the physical world and our interactions with it.

This paper starts from the above considerations and presents the design and implementation of a system for tuple-based coordination in physical environments enriched with RFID tags [12]. The key idea is that our everyday environments will be soon pervaded by RIFD-tagged objects, each with some amount of accessible digital memory, in which it will be possible to store and retrieve data via wireless access. On this base, it will be possible to exploit these physically embedded memory areas to enforce tuple-based coordination. Simply, agents in an environment (whether they are robots or humans carrying on a mobile device) can "access" such digital memory using tuple-based primitives and, in this way, retrieve useful contextual information and coordinate with each other.

The proposed approach – being flexible, easy to deploy and to use, and very cheap – has the potential to be very useful in a variety of application scenarios. Also, our approach clearly distinguishes from related work in the area. On the one hand, RFID technology has been exploited so far only for reading pre-existing information on tagged objects and to exploit this information for logistics purposes [2] or to detect activities in an environment [10]. On the other hand, systems for pervasive tuple-based coordination proposed so far exploit costlier and more difficult to deploy technologies than RFID [3].

The remainder of this paper is organized as follow. Section 2 briefly introduces RFID technology and discusses related work in the area. Section 3 presents our system for RFID tuple spaces. Section 4 introduces an application example to clarify the use of our system. Section 5 concludes, by discussing the current limitations of our systems and the avenues for future research.

2. RFID TECHNOLOGY
In this section, we briefly introduce RFID technology, and then discuss related work in the area, i.e., how RFID technology has been used so far to improve interactions with the physical world.

2.1 Overview
Advances in miniaturization and manufacturing have yielded silicon-based postage-stamp-sized radio transceivers, called Radio Frequency Identification (RFID) tags that can be attached unobtrusively to objects as small as a pen (Figure 1a). Each tag is marked with a unique identifier and is provided with a tiny re-writable memory, up to some KB for advanced models (512bit in our implementation). The memory of tags, organized in cells, can be exploited to store information in a persistent way. Tags can be purchased off the shelf and, being battery-free, can withstand day-to-day use for years.

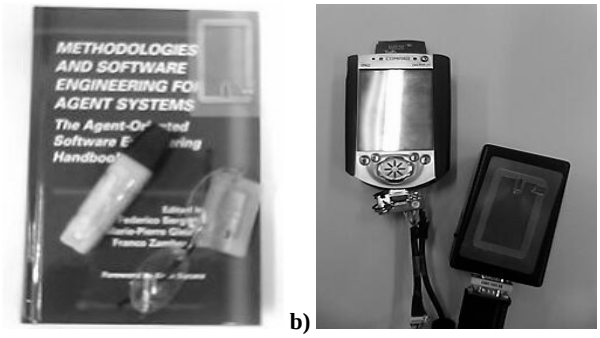
Suitable devices, called RFID readers, can be can be used to access the memory of RFID tags for both read and write operations. A RFID reader can be interfaced with any computer-based device, even portable (Figure 1b). RFID readers access to RFID tags wirelessly. When in need of reading or writing a tag,

SAC'06, April, 23-27, 2006, Dijon, France.

an antenna on the RFID reader emits a radio signal that has the twofold function of commanding the tag (e.g., "I want to read your unique identifier", "I want to write the byte FF in the third memory cell") and of bringing energy to it. When receiving a signal, a tag takes power from it, activates itself and responds or stores data according to the command embedded in the signal. RFID readers divide into short- and long-range depending on the distance within which they can access RFID tags. Such ranges vary from a few centimeters up to some meters.

Currently, RFID tags are in widespread use as anti-theft technology in shops and for the production of personal ID cards. Also, a growing number of business and logistics information systems are exploiting RFID tags for automating the identification and the tracking of products and merchandises [2]. In these systems, each product in a warehouse or in a production line is uniquely identified by a tag, possibly including some description of the product. The automatic gathering of the data in the tag and its proper integration into an information system allows for better and cheaper management operations.

The trends indicate that, in a few years, most household objects and furniture will be RFID-tagged before purchase. The increasing diffusion of the technology will make it cheaper and cheaper. Moreover, handheld devices will be more and more provided with RFID reading and writing capabilities (e.g., the Nokia 5140 phone). The future pervasiveness of RFID technology, together with its low cost, makes it an interesting option to explore for pervasive computing scenarios.



a) b)

Figure 1. (a) Some tagged objects. (b) An RFID reader connected to a PDA.

2.2 RFID in Pervasive Computing

Several proposals in pervasive computing exploit RFID tags as a simple and cheap way to enforce context-awareness.

The system described in [11] assumes that readers are static, embedded in the environment, and connected to a fixed network infrastructure, while tags are attached to objects and mobile devices. This can be used by the network to recognize the presence of specific tagged objects or tagged PDAs into a specific room/location, and to contextualize activities of applications accordingly: if the RFID reader in a room can read a specific tag, this means that the corresponding object is in the proximities.

Several other proposals, as well as ours, consider that more flexibility can be achieved by having mobile devices themselves integrated with a RFID reader, thus having the capability of accessing RFID tags around, as sorts of digital contextual information stores. However, rather than considering the possibility of storing new information in RFID tags and enforcing coordination through them (as we do), most approaches exploit RFID tags only for reading pre-existent environmental/contextual information. For instance, the system described in [7] proposes associating location information with tags (e.g., "I am the tag of the living room") that can be read by mobile robots carrying on a RFID reader to roughly localize themselves. The system described in [10] exploits RFID tags for inferring information about contextual activity in an environment. Users are assumed to wear an RFID reader connected with a Wi-Fi portable device so that, when the user moves and acts in the environment, the type and the sequence of tags read by the reader can suggest what the user is doing.

None of the above proposals exploit RFID tags as environmental memories for writing application-specific information. The middleware architecture proposed in [5] (not implemented at the time of writing), however, shares with our work the motivation of defining a general framework for exploiting RFID tags for both reading and writing application-specification information. There, the authors propose a sort of content-based publish/subscribe mechanisms through which applications can select to which type of tagged data they are interested in, while the middleware service takes care of controlling RFID readers in reading/writing data and in delivering it to applications accordingly. Such proposal, as ours, enforces associative access to tag data. However, it considers multiple RFID readers embedded in an environment and controlled by a single application agent. Our proposal, instead, assumes that each RFID reader is attached to a single device, and that application agents running on such device can directly control the reader.

In a previous work [8], we have exploited RFID tags distributed in an indoor environment as a memory infrastructure in which to store pheromone paths for the sake of enforcing pervasive stigmergic (ant-like) coordination. However, in that work, we neither have implemented any specific tuple space abstraction, nor defined a general API to access RFID tags. Conversely, the tuple space system presented in this paper can be indeed exploited also to deploy distributed pheromone paths.

A proposal that, although not exploiting RFID technology, has some relation with, is the TinyLime middleware [3]. TinyLime proposes accessing the environmental data collected by a sensor network via an associative tuple-based mechanisms: when a user with a mobile device "walks-through" a network of distributed sensors, all the data collected by the in-range sensors automatically feeds a local tuple space of the mobile device, which thus can perceive sensorial data collected by sensors simply by reading in the local tuple space. Thus, as far as reading environmental data is involved, TinyLime is analogous to our proposal. However, TinyLime assumes the presence of computer-based sensors, much more costly than RFID tags and exhibiting battery exhaustion problems. Also, TinyLime misses the possibility of exploiting sensors to store (write) information and to enforce coordination between mobile devices.

3. RFID TUPLE SPACES

In this section, we give a general overview of our system, detail its application programming interface, and provide some technical remarks on its current implementation.

3.1 Overview

We assume a scenario in which the environment is densely populated by RFID tags, possibly attached to doors, walls, furniture, objects of any kind, acting as the distributed digital memory in which tuples are stored. For what said in Subsection 2.1., the presence of tags will be more and more pervasive, mostly avoiding the need of a priori deployment effort. In any case, enriching an environment with tags is a simple operation, reducing at sticking them around as needed. Also, our system considers the possibility that an environment can be enriched by tags on-the-fly, whenever needed for application purposes. In other words, if there is need of environmental memory for users to store information, users can dynamically release a tag in the environment and immediately use it (e.g., storing a tuple there).

Users of the system can be humans carrying on a PDA connected to a RFID reader (as from Figure 1b) or autonomous robots again connected to a RFID reader. Whatever the case, users can roam in the environment and there can exploit the environmental memory of RFID tags. In particular, application agents running on the PDA or on the robots can control their own RFID reader to access the environmental memory via simple tuple space operations. These operations provide agents the capability of storing new tuples in the environment and of reading/extracting tuples in an associative way. Also, blocking read/extract operations and asynchronous notification mechanisms enable application agents on different devices to coordinated and synchronize their activities with each other through the environmental memory. In any case, we emphasize that environmental memory may not necessarily be the only coordination mean. For instance, Wi-Fi PDAs or robot could also directly coordinate with each other via TCP/IP. But this is not our focus here.

From the viewpoint of application agents, the perception of the environment is that of a normal tuple space. From a given position in an environment, an agent has access to a locally-bounded set of RFID tags, depending on the range of the associated RFID reader (see Figure 2). Thus, performing input tuple space operations implies extracting tuples by applying the associative access mechanism to the aggregate memory of in-range tags. Analogously, inserting a tuple in the tuple space implies storing it in some available memory cells of one of the in-range tags (although we also provide the possibility of storing tuples in specific tags).

It is worth emphasizing that in our system, unlike in most of tuple-based middleware, there is no such concept as a single global tuple space [1] or as a multiplicity of localized tuple spaces [9]. In our approach, tuples are spread everywhere in the physical space, without being at all pre-organized in logical containers and rather defining a sort of "continuum" of tuples. The concept of tuple space is subjective, depending on the current position of an agent and on the range of its RFID reader. The same agent perceives different information in different physical position, while two agents in the same position may perceive different information depending on the reading range. Clearly, this implies that coordination between two agent has to be necessarily contextual, i.e., spatial. Two agents can synchronize and coordinate only when they access (possibly at different times) to intersecting sets of tuples, that is, when they act in close position of the spatial environment.

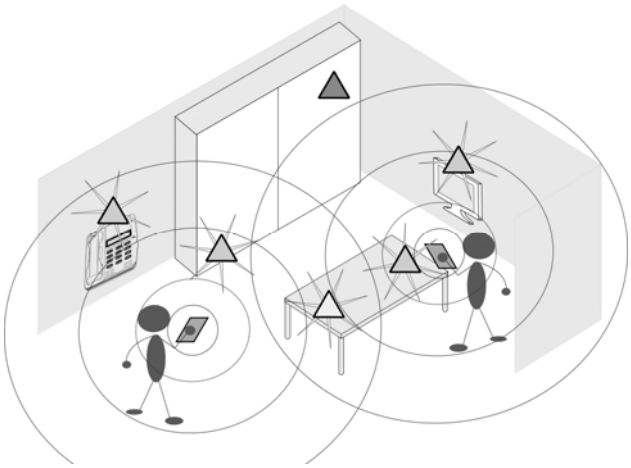


Figure 2. Two users (or of the same user at different positions) accessing different set of tags in an RFID enriched environment.

3.2 Programming Interface

The API of our system defines Linda-like Java primitives to write tuples in the environment, read or extract tuples, as well as primitives to subscribe/unsubscribe to events. The usage of the primitives does not require any a priori bounding to a specific tuple space. Instead, an agent invokes a primitive and this primitive will automatically act on the logical tuple space defined by the currently in-range RFID tags.

Despite the Linda common ground, some important differences have also been necessary to deal with the extremely limited storage capability of RFID tags.

The main difference from the "standard" Linda model, is about the definition of a tuple. In our implementation, a tuple is defined by means of a simple name and value pair. Given the limited storage capacity of tags, names and values are coded with just 1 byte each. An application-dependent ontology (i.e. codebook) provides the right correspondence between high-level objects and low-level codes. For example, if an agent wants to create the tuple ("apple", "red"), it will look for the code of the strings "apple" and "red" in the ontology (i.e. "apple"=0F, "red"=10), and then it will actually create (0F,10). Once a specific ontology has been selected, the encode and decode operations are invisible from the application point of view. Templates are analogous to tuples but the value part can be left unspecified (null) to represent wild-cards. Both tuples and templates are encoded in a data-structure called *TagEntry*.

Taking inspiration from JavaSpace terminology we defined the *write* primitive to insert a tuple in the environment, as follows:

```
boolean write(TagEntry[] tuple, TagEntry[] template);
```

In our implementation, this method can write more than one tuple at once (it writes an array of tuples) and it returns a boolean indicating whether the operation was successful or not. This capability of writing multiple tuples gives to the application developer the illusion of being able to manage complex tuples consisting of several name and value pairs:

```
/* create a tuple*/
TagEntry[] tuple = new TagEntry[2];
tuple[0] = new TagEntry("apple", "red");
tuple[1] = new TagEntry("cost", "10");
// write the tuple in the RFID tag
write(tuple, null);
```

Another remarkable difference with respect to standard Linda operations is the introduction of a template matching process also in the *write* operation. Since in a given environment there may be several in-range tuple spaces (i.e. RFID tags), it may be useful to impose some constraints on where to actually write a tuple. To this end, the second parameter of the *write* method is an array of templates (name-value pairs with wild-cards). The write operation first selects among in-range RFID tags, those storing tuples that match *all* the templates in the array (for each template there must be at least one matching tuple). Then, it writes the new tuple in *one* of the matching tags by a random selection:

```
/* create a tuple*/
TagEntry[] tuple = new TagEntry[2];
tuple[0] = new TagEntry("apple", "red");
tuple[1] = new TagEntry("cost", "10");
/* create a template */
TagEntry[] template = new TagEntry[1];
template[0] = new TagEntry("shop","open");
// this writes the tuple in one tag picked randomly
write(tuple, null);
/* this writes the tuple in one tag picked randomly among
those already containing the tuple "shop=open" */
write(tuple, template);
```

The other Linda methods follow rather simply from these considerations. The *read* and *take* primitives are defined to read or extract a tuple matching a given template from the environment:

```
TagEntry[] read(TagEntry[] template, TagEntry[] tag, int lease);

TagEntry[] take(TagEntry[] template, TagEntry[] tag, int lease);
```

In these methods there are two template matching processes. The first (second parameter) selects some matching tuple spaces (i.e. tags) among the in-range ones, on the basis of their content, in a process similar to the one described above. The second template-matching process (first parameter) selects some tuples from the previously selected tuple spaces. At the end, one of the resulting tuples is picked randomly and either read of taken.

Finally, we implemented a *subscribe* primitive to have agents notified whenever a tuple matching a subscription template enters the environment (or, which is the same from the agent viewpoint, when the agent enters a region of the environment in which the tuple can be found). The *subscribe* method returns an unique identifier to be possibly used later to remove that subscription with the *unsubscribe* method:

```
int subscribe(TagEntry[] tuple, TagEntry[] tag, Listener listener);

void unsubscribe(int subid);
```

In conclusion, we want to emphasize that some of the above differences from the standard Linda model are likely to disappear when tags with more storage capacity will be available. For example, we expect that both the need to have an ontology (codebook) mapping java objects to bytes, and the constraint of having a tuple defined in terms of a name-value pair will be removed when it will be possible to store in tags complete serialized objects.

3.3 Implementation

From the hardware viewpoint, in our implementation, each application agent runs on a HP IPAQ 36xx PDA, running Familiar Linux 0.72 and J2ME (CVM – Personal Profile) and provided with a WLAN card and an Inside mid-range RFID reader. Thus, each agent can be carried around and it can connect to different tags deployed in the environment. In particular, we employed ISO15693 RFID tags, each with a storage capacity of 512 bits. These tags communicate at the 13,56MHz frequency and the reader can interact with tags within 1,5 meters. The information being read is passed to the PDA via a serial cable where it is processed. In addition, a mobile robot has been realized by installing one PDA connected to a RFID reader onboard of a Lego Mindstorms robot. The IPAQ runs an agent controlling both the RFID reader and the robot microprocessor.

From the software viewpoint, we have implemented a layered software architecture (see Figure 3).

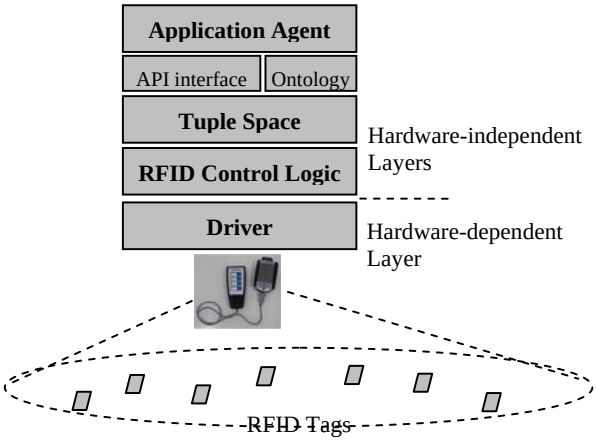


Figure 3. Software Architecture

The bottom level consists in the RFID reader device-driver. This driver has been realized in terms of a dynamic link library that exports a number of methods to control the reader. In particular, we implemented methods to read and write a specific byte in a specific tag. This is the only layer in our architecture that is hardware dependent.

On top of the driver, we realized a RFID control logic layer. Since RFID communication is error-prone (it may happen that in-range tags fails to respond), we developed some mechanisms to iterate operations, reading tags multiple times to increase probability of reading all tags, and merge the results accordingly.

The tuple space layer is the core of our tuple space implementation. It is in charge of performing important operations such as: pattern matching and notifying agents upon triggering conditions (e.g. with regard to event subscriptions and blocking read). For example, upon the execution of a blocking read, this layer remains active and ready to notify the application

layer upon new tuples being inserted in the tags around or new tags coming in-range due to device movements.

The API Interface and Ontology layer provides the Linda-like methods described above and the (application-specific) ontology to map high-level objects in low-level byte-values to be stored in the tags. This ontology is integrated with the *TagEntry* constructor (see above) to hide the low-level representation from the application point of view.

The top layer of our architecture is the application agent. Although nothing would prevent from running more than one agent on the same platform that coordinate using the available tags, our standard application scenario involves – rather naturally – multiple devices each with a single agent.

3.4 Reliability Considerations

In general, the act of reading and writing RFID tags is something that exhibits high degrees of uncertainty, for various reasons [4]: the more a tag is far from the reader, the more the probability it is not perceived; the orientation of the tag with respect to the antenna influences the capability of the reader of perceiving it, and even a very close tag may happen not to be readable; metal object in the proximity of a tag may prevent a tag from being activated. For these reasons, in our system as well as in most systems based on RFID tags, an RFID control layer is introduced. Nevertheless, there is not any guarantee that a tag in the range of a reader will be properly read/written by the reader.

In our system, the impact of the above problem is intrinsically limited by the adoption of the tuple space coordination model. The uncertainty in reading tags and the fact that some existing tags may stay undetected (or, which is the same, the fact that some existing matching tuple does not end up in an actual match) does not generally cause, from the viewpoint of the tuple-based coordination model, any application-level critical error. The fact that, among multiple matching tuples existing in the environment, only a few of them may be detected at a given time by a reader, can be perceived by an application agent as part of that non-deterministic process of tuple selection which is promoted by tuple-based coordination. Of course, for specific critical applications this may be unsatisfactory, calling for, e.g., tuple replication over multiple tags to increase reliability.

Getting to performance considerations, the limited radio frequency at which tag readers have to operate together with the above identified reliability problems bounds the number of distinct tags that can be read in a timeframe to about 20-100 per second, depending on the specific technology and on the characteristics of the environment. Given the kind of application we consider (interactive and characterized by slow mobility), such numbers appears satisfactory, although a deeper analysis of performance issues may be needed in any case.

4. APPLICATION EXAMPLE

The implementation of tuple spaces within RFID tags enables to exploit the benefits of tuple-based coordination in a wide number of pervasive computing scenarios.

As a first example, tuples written in the environment can be used to help users (as well as robots) in getting aware of what's in the environment more than their natural and artificial senses can do, as it can be the case with objects annotated with usage instructions. As another example, tuples spread in the environment could enable a group of agents (both humans and robots) to coordinate their respective movements, e.g., for distributed exploration: agents can mark visited areas via proper RFID tuples, and other agents could decide to explore a specific area if there are not any tuples pointing in that direction (the area is unexplored). More in general, one can consider using RFID tuple-based coordination to enforce distributed workflow management. RFID tags associated to items to be processed could store a tuple identifying the operations that the item undertook. Workers with RFID readers could simply read the state of the item and process it further. This could be employed in traditional manufacturing as well as in more mundane domestic workflows.

To better exemplify the usage of our RFID tuple-based system, let us focus on a first-aid response scenario after an earthquake. First responders coming to the disaster area have the primary need to coordinate their search for injured people, but cannot rely at all on wired or wireless network infrastructures. Accordingly, they could use RFID tags likely to be present in the crumbled environment or leave new tags to coordinate efficiently.

When looking for people, each rescuer could instruct his PDA to simply write the tuple (rescuer=my_id) in the tags around – to indicate that the area has already been explored. The PDA can then be programmed to read periodically the tags around and notify to the rescuer if the area had been visited before by another member of the rescue team (see code below).

```
while(true) {
 /* read tag around */
 TagEntry[] template = new TagEnrty[1];
 template[0] = new TagEntry(rescuer);
 TagEntry[] ts = read(template, null, 0); // non blocking
 if(ts contains other rescuer IDs)
  alert("The area has already been explored");
  /* write rescuer ID */
 TagEntry[] tuple = new TagEnrty[1];
 TagEntry[0]= new TagEntry(rescuer, my_id);
 write(tuple, null);
}
```

At the same time, rescuers could leave useful context-information related to environmental conditions. Such information could be valuable to other team members coming later to the area. For example, first rescuers could leave tuples indicating if the area is suitable for ambulance crossing. Ambulance equipped with RFID readers can read such tuples and alert their drivers accordingly.

Tuple-based coordination and RFID tags could also support the workflow in medical operations. First responders, after visiting some injured people (see Figure 4-a), could stick RFID tags to them and write there tuples with relevant trauma care information (see Figure 4-b). Adopting the proposed infrastructure, the code to deploy such tuples becomes rather trivial:

```
/* create a tuple storing: medicated = true, to_be_carried = true, illness = medium*/
TagEntry[] tuple = new TagEnrty[3];
TagEntry[0] = new TagEntry(medicated,true);
TagEntry[1] = new TagEntry(to_be_carried,true);
TagEntry[2] = new TagEntry(illness, medium);
// write the tuple in the patient RFID tag
write(tuple, null);
```

After that, first responders, could move further visiting other persons. In the mean time, nearby paramedics and emergency medical technicians could care for already visited patients by taking advantage of the information stored in their RFID tags (see Figure 4-c), as in the code below:

```
/* create a template with: to_be_carried = true,*/
TagEntry[] template = new TagEnrty[1];
template[0] = new TagEntry(to_be_carried,true);
/* start a blocking read to find a suitable patient */
read(template, null, true);
/* notify that a patient to be carried has been found */
…
```

This could notably reduce latencies and delays and allow to care for several patients at once in parallel. After this first stage, some patients would be carried in the ambulance. During the journey to the hospital, the tuples stored in the RFID tags could be integrated with additional information and be sent wirelessly to the hospital. This would allow doctors there to save time by setting-up things in advance (e.g., preparing surgery rooms) to receive the incoming people more quickly and more efficiently.

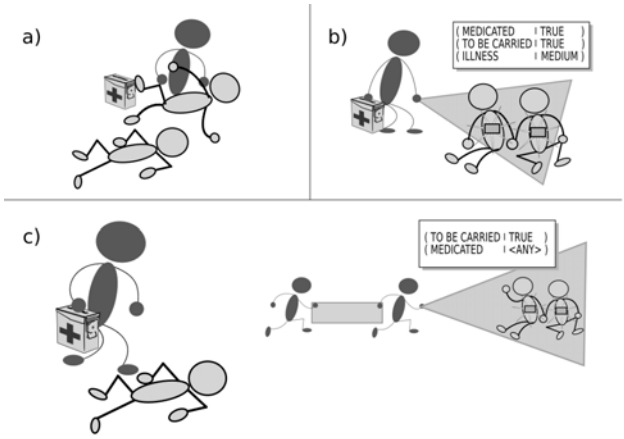


Figure 4. Tuple-based coordination and RFID in a first-aid scenario. (a) first responder, after providing first-aid to injured people, (b) store in RFID tags attached to them relevant information about their conditions. Then, (c) paramedics can read those RFID tags to get aware of and treat injured people quickly and efficiently.

5. CONCLUSIONS AND FUTURE WORK

We have presented the design and implementation of a system that, by exploiting the physically distributed memory of RFID-tags, enables to enforce pervasive tuple-based coordination activities. We think this is a promising solution to properly integrate RFID technology into modern pervasive computing scenarios, and possibly to seamlessly integrate "traditional" digital data with "physical" RFID data within a homogeneous coordination model.

We are aware of a number of limitations of our proposed system that may undermine its general usability. First, RFID tags introduces privacy and security issues (anyone can read and modify what's in the tags), calling for proper solutions both at the hardware and at the software level. Second, specific applications may require higher degrees of reliability that we can currently provide, possibly coupled with transactional mechanisms. Both these issues are currently being investigated.

6. ACKNOLEDGEMENTS

Work supported by the Italian MIUR-CNR in the context of the project "IS-MANET".

7. REFERENCES

[1] S. Ahuja, N. Carriero, D. Gelernter, "LINDA and Friends", *IEEE Computer*, 19(8):26-34, Aug. 1986.

[2] C. Bornhovd, T. Lin, S. Haller, J. Schaper, "Integrating Automatic Data Acquisition with Business Processes: Experiences with SAP's Auto-ID Infrastructure", *30th International Conference on Very Large Databases*, IEEE CS Press, pp. 1182-1188, 2004.

[3] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco, "TinyLime: Bridging Mobile and Sensor Networks through Middleware", *3rd IEEE International Conference on Pervasive Computing and Communications*, IEEE CS Press, pp. 61-72, March 2005.

[4] K. P. Fishkin, B. Jiang, M. Philipose, S. Roy, "I Sense a Disturbance in the Field: Unobstrusive Detection of Interactions with RFID Tagged Object", *UBICOMP 2004 Conference*, pp. 268-282, LNCS, Sept. 2004.

[5] C. Floerkemeier, M. Lampe, "RFID middleware design - addressing application requirements and RFID", *Technical Report ETH Zurich*, Pervasive Computing Group, 2005.

[6] D. Gelernter and N. Carriero, "Coordination Languages and Their Significance", *Communications of the ACM*, 35(2):96–107 (1992).

[7] V. Kulyukin, C. Gharpure, J. Nicholson, S. Pavithran, "RFID in Robot-Assisted Indoor Navigation for Visually Impaired", *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE Press, 2004.

[8] M. Mamei, F. Zambonelli, "Spreading Pheromones in Everyday Environments Through RFID Technology", *2nd IEEE Symposium on Swarm Intelligence*, IEEE Press, pp. 281-288, June 2005.

[9] A. Omicini, F. Zambonelli, "Coordination for Internet Application Development", *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999.

[10] M. Philipose, K. P. Fishkin, M. Perkwitz, D. J. Patterson, D. Fox, H. Kautz, D. Hahnel, "Inferring Activities from Interactions with Objects", *IEEE Pervasive Computing*, 3(4):10-17, Oct.-Dec. 2004.

[11] I. Satoh, "A Location Model for Pervasive Computing Environments", *3rd International Conference on Pervasive Computing and Communications*, IEEE CS Press, pp. 215-224, March 2005.

[12] V. Stanford, "Pervasive Computing goes the Last Hundred Feet with RFID Systems", *IEEE Pervasive Computing*, 2(2):9-14, April-June 2003.